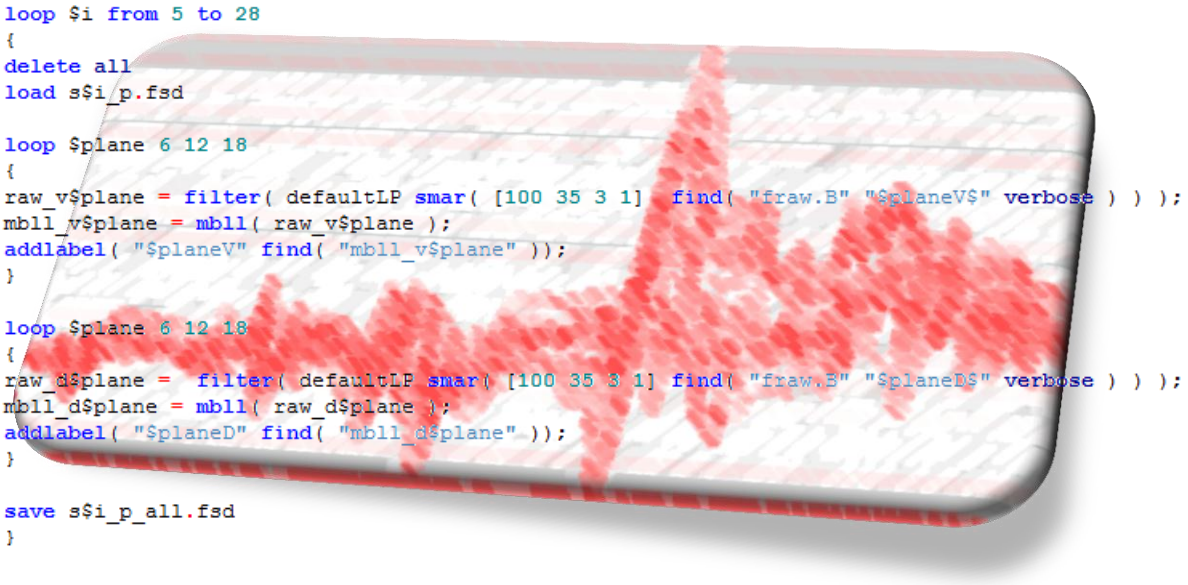
fnirSoft Scripting

A Quick Start Guide

12/27/2011
Drexel University
Hasan Ayaz


loop $i from 5 to 28
{
delete all
load s$i_p.fsd
loop $plane 6 12 18
{
raw_v$plane = filter(defaultLP smar([100 35 3 1] find("fraw.B" "$planeV$" verbose)));
mbll_v$plane = mbll(raw_v$plane);
addlabel("$planeV" find("mbll_v$plane"));
}
loop $plane 6 12 18
{
raw_d$plane = filter(defaultLP smar([100 35 3 1] find("fraw.B" "$planeD$" verbose)));
mbll_d$plane = mbll(raw_d$plane);
addlabel("$planeD" find("mbll_d$plane"));
}
save s$i_p_all.fsd
}

Please refer to fnirSoft in your publications with the following:

Ayaz, H. (2010). "*Functional Near Infrared Spectroscopy based Brain Computer Interface".* PhD Thesis, Drexel University, Philadelphia, PA.

Disclaimer

THIS SOFTWARE IS PROVIDED ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table of Contents

Introduction 5
Overview 5
Quick Start 6
Hello World! 6
Arithmetic / Trigonometry 6
Creating and Using Variables 7
Creating a vector variable 7
Using Auto-complete 8
Editor Tool 9
Loops 10
Nested loops 10
Conditionals 11
Debugging 11
Current directory 12
Common usage patterns 13
Using find command 13
Calculating Averages 14
Load/Save Variables 16
Applying FIR Filter 16
Calculating Oxygenation 18
Standardizing 19
Splitting Variables 19
Exporting variables to Matlab 20
Exporting variables to Text Files 20
Naming Convention of Commands 21
List of Commands 22
Math/Arithmetic 22
Utilities/Tools 22

Statistical Comparison 23
Signal Processing - Temporal 23
Signal Processing - Spatial 24
Signal Processing - Cell by Cell 25
Signal Processing - Misc/Tools 25

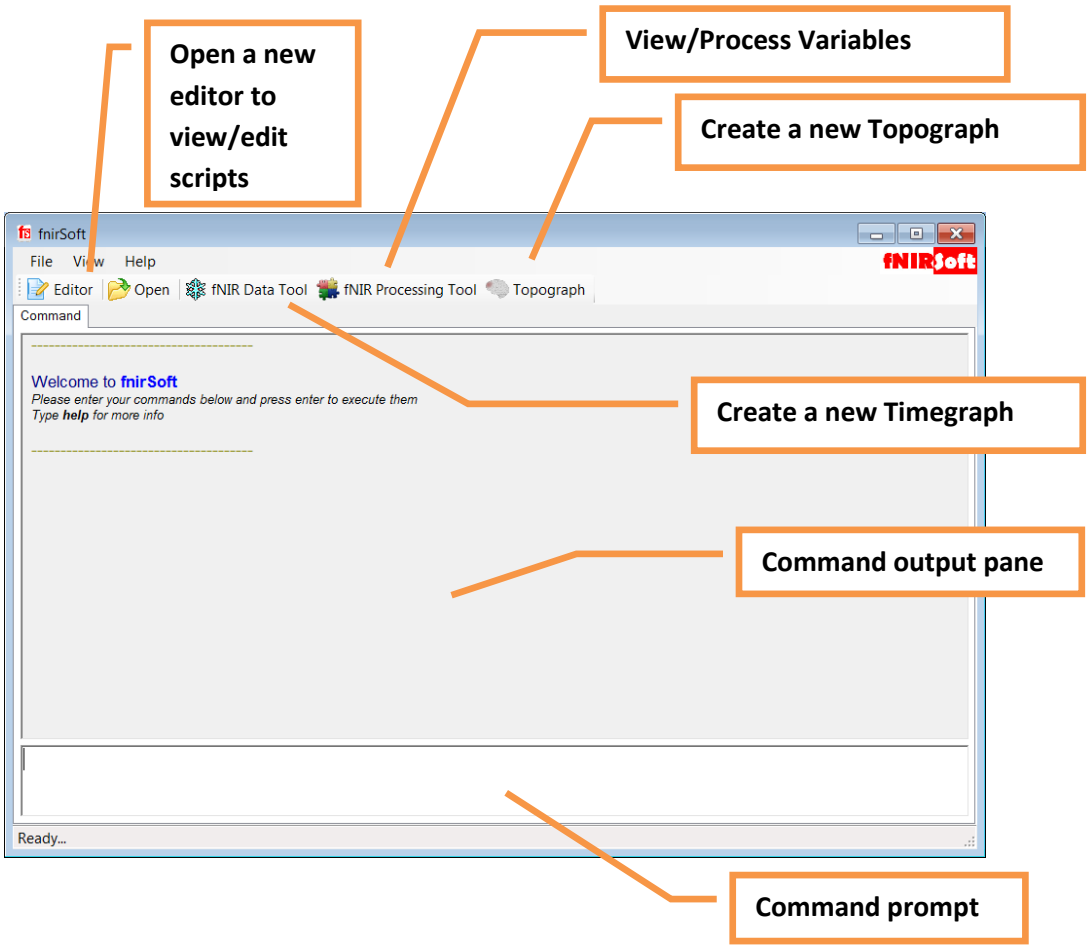
Introduction

fnirSoft (fS) is a stand-alone software package designed to process, analyze and visualize functional near infrared (fNIR) spectroscopy signals through both graphical user interface and/or scripting. This document provides an introductory tutorial for fnirSoft scripting (programming).

fS script is a data-driven functional programming language designed to do more with less code and be both readable by humans and computers. The idea is to write simple and clean text without cryptic text and easily manage and automate data processing for scientific computing. fS Script is also aimed to help repeatable research by standardizing functional neuroimaging signal processing steps and help establishing conventions that can be repeated over time on different datasets.

Overview

Below is the main window of fnirSoft with common user elements identified. For more information please refer to fnirSoft User Manual (2011).



fnirSoft has a scripting engine and commands written to the “command prompt” at the bottom of the window are executed by pressing enter. Algebraic expressions can be evaluated. Overview of the syntax and step by step tutorials will be provided in the next chapters.

Quick Start

Scripts are series of commands written in text. They can be executed either by typing at command prompt or by saving them in separate files and executing the file.

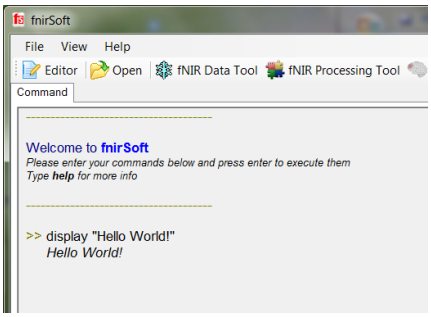
Parentheses are used to prioritize processing and identify parameters for a function.

Space is the basic separator between functions, names and everything…

Hello World!

Here’s your first fnirSoft program. Type in the following at command prompt and hit enter. The code will be repeated at the command output pane and output will be displayed below it as shown below.

```
display "Hello World!"
```


fnirSoft
File View Help
Editor Open fNIR Data Tool fNIR Processing Tool
Command
Welcome to fnirSoft
Please enter your commands below and press enter to execute them
Type help for more info
>> display "Hello World!"
Hello World!

Arithmetic / Trigonometry

Binary arithmetic operators + (addition), - (subtraction), * (multiplication), / (division); and unary operators ^ (power), ‘ (transpose) are available.

```
12 + 5
```

```
>> 12 + 5
   17
```

```
12 - 5
```

```
>> 12 - 5
   7
```

```
0.5 * log ( exp ( 10 ) ) + ( 16.2 * 5 - 1)
```

```
>> 0.5 * log( exp( 10 ) ) + ( 16.2 * 5 - 1)
   85
```

```
sin ( pi / 2 )
```

```
>> sin( pi / 2 )
   1
```

```
atan ( tan( pi / 2 ) ) / pi
```

```
>> atan ( tan( pi / 2 ) ) / pi
   0.5
```

TIP

If you add ';' (semicolon) after the command, no output will be typed in the command output pane,

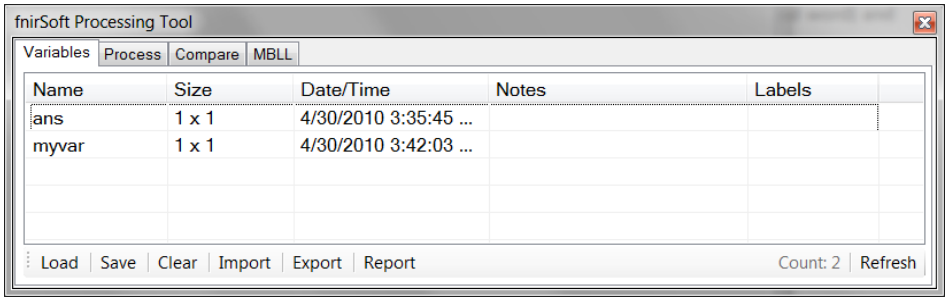
Creating and Using Variables

To create a new variable, just type a valid name (that starts with a letter and not a reserved word) and used '=' to assign value to the variable.

```
myvar = 5
```

```
>> myvar = 5
   myvar =
   5
```

With this single assignment, a new variable called 'myvar' is created. A number is assigned as value and the size of the variable is 1x1. All available variables can be seen at the 'Variables' tab of 'fNIR Processing Tool' that can be opened by typing '**variables**' at the command pane and hitting enter or, by using the toolbar button at the main window (See figure in page 3).



A variable can be single value, a vector an array. After creating the variable, the name can be used in subsequent commands.

```
myvar + 2
```

```
>> myvar + 2
   7
```

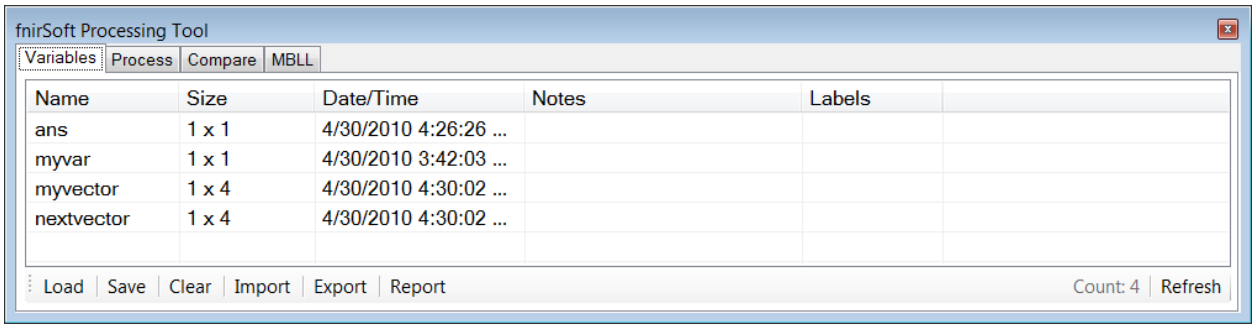
Creating a vector variable

To create a vector, type values separated by space between squared paranthesis: '[' and ']' as shown below.

```
myvector = [ 1 2 3 4];

nextvector = myvector * 2
```

```
>> myvector = [ 1 2 3 4];nextvector = myvector * 2
   nextvector =
   2 4 6 8
```

After the two commands are executed, the new variables are also available in the list.



TIP

Use '**list vars**' command to list variables with sizes within the command pane.

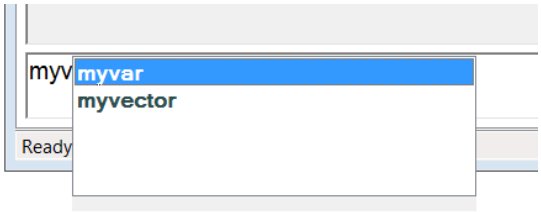
```
>> list vars
   ans          1x1
   myvar        1x1
   myvector     1x4
   nextvector   1x4

   Total number: 4
```

Using Auto-complete

When typing variable names or command names, auto-complete tool can be used. After you type a few letters, hit 'tab' button (on the keyboard), this will pop-up a list that displays commands and variables that include that partial text. You can hit enter to select one (that will insert the selected) or hit 'Escape' button (on the keyboard) to cancel.

For example, type 'myv' three characters and then tab button on the keyboard, a popup-window with the following names (variables created in previous examples) will be shown as candidates.

myv***<tab>***



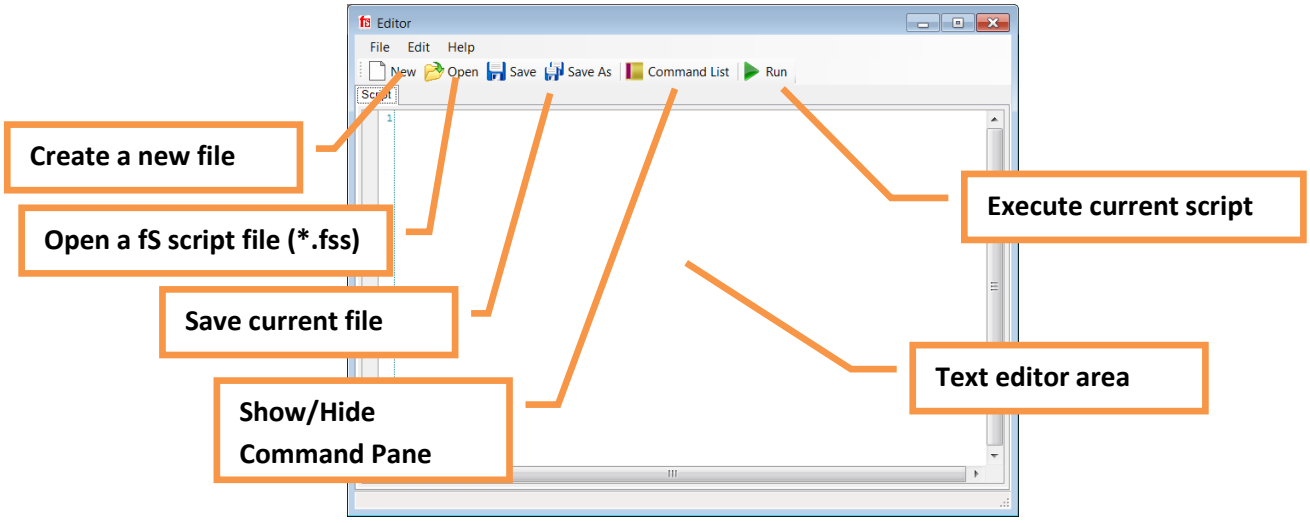
Note that, if there are no alternative (only one option) with entered partial text, the list will not be shown and the only option will be entered immediately.

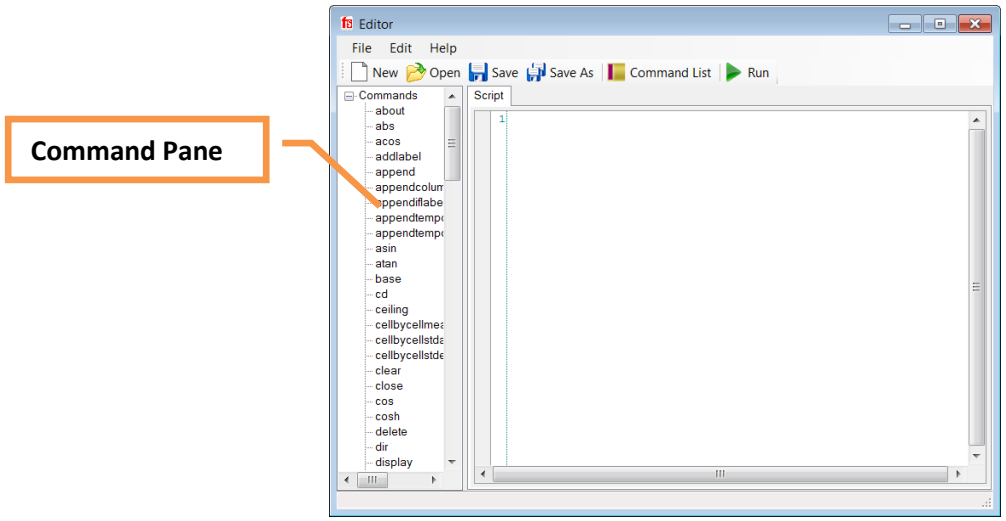
TIP

Use '**up/down**' cursor keys (on the keyboard) to navigate between commands entered previously.

Editor Tool

Editor is fS Script editing tool that provides loading/saving/executing fS Programs. Editors can be opened by clicking the 'Editor' button at the toolbar of main window or just typing '**editor**' at the command prompt. Below is an empty editor window.

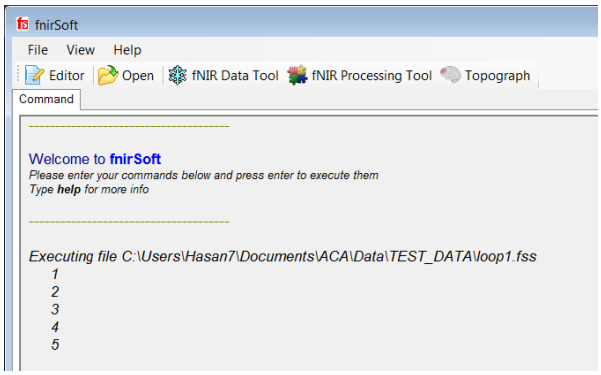




Loops

Iterations and repetitions are critical in performing same task on large amount of data. To create a loop, use "**loop"** keyword followed by a **helper variable** name (i.e. any valid variable name that starts with $ sign) and use, "**from**" and "**to**" keywords to indicate start and end counts for the helper variable to go through. After typing the code, hit 'Run' button (at the editor toolbar) to execute the script. Output will be shown at the command output pane of the main window.

```
// count from 1 to 5

loop $i from 1 to 5
{
    display "$i"
}
```




```
// count from 1 to 5 with increments of 2

loop $i from 1 to 5 step 2
{
    display "$i"
}
```

```
Executing file C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\loop2.fss
1
3
5
```

```
// count free-form..

loop $i 5 2 3 1
{
    display "$i"
}
```

```
Executing file C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\loop3.fss
5
2
3
1
```

Nested loops

Loops within loops can be created, below example code has two nested-loops and the output indicates the order of execution.

```
// nested loops

loop $i from 1 to 3
{

    loop $j from 10 to 7
    {
        display "$i $j"
    }

}
```

```
Executing file C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\loop4.fss
1 10
1 9
1 8
1 7
2 10
2 9
2 8
2 7
3 10
3 9
3 8
3 7
```

Conditionals

If-else statements can be used to create operation branches. The syntax is as follows.

If *logical-statement*
    Operation1
else
    Operation2

The logical statement is composed of two values separated by a comparison operator ('<', '>' or '='). Else statement is optional. Both operation1 and operation2 should be single operations that is on single line without any ';'.

```
1
2   testvar = 1
3
4   if testvar  > 0
5   display "OK"
6   else
7    display "NOT OK!"
8
9
10  display "The end!.."
11
12
```

```
Executing file C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\conditional1.fss
    testvar =
    1

    OK
    The end!..
```

Debugging

When working with long script files, a way to identify problematic code sections is important. To see an example, consider the following example, in which, somewhere in the code, a variable name is used without defining it. When the code is executed, the line number and the problem are reported.

```
2   // nested loops
3
4   loop $i from 1 to 3
5   {
6
7       loop $j from 10 to 7
8       {
9           display "$i $j"
10          a= somevariable
11      }
12
13  }
14
```

```
Executing file C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\loop4.fss
    1 10
    No variable provided set after '='
    Line Number: 10
```

Note that, in editor, line numbers are also provided on the left of each line.

> **TIP**
>
> Use '**delete all**' command to delete all variables in memory.
>
> Or use "**delete <variable name>**" to delete specific variables.

Current directory

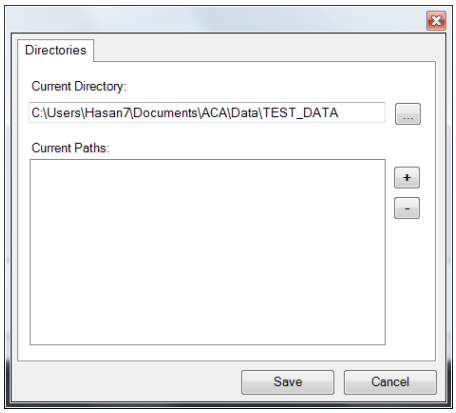
Current directory is a setting that can be changed. When using commands to do file operations (read/write files) if only file names are provided, they are expected to be in the current directory.

To check current directory, use 'pwd' command.

```
pwd
```

```
>> pwd
   C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA
```

Alternatively, current directory can be checked and changed by clicking "View > Settings" from the top menu of the main window.



Current directory can be edited or the "…" button can be used to select a folder. When a folder is saved, the setting will persist beyond a restart.

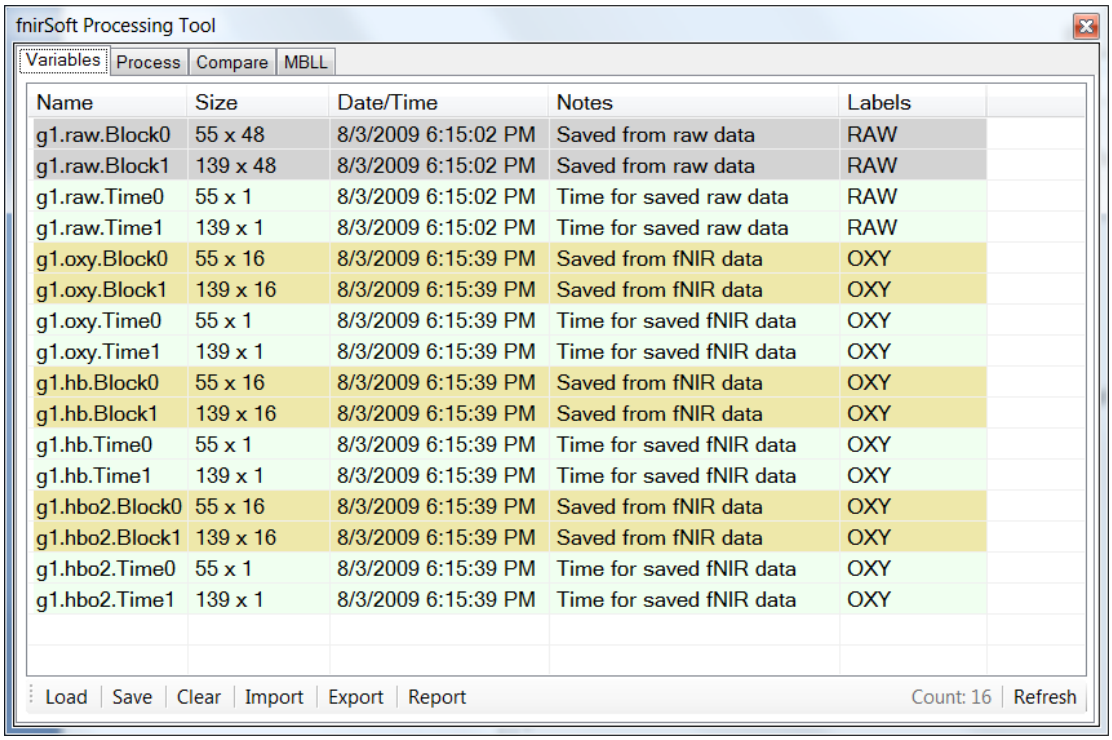
To temporarily change the current directory, use 'cd' command.

```
cd "C:\Users\Hasan7\Documents\ACA\Data"
```

```
>> cd "C:\Users\Hasan7\Documents\ACA\Data"
   C:\Users\Hasan7\Documents\ACA\Data
```

Common usage patterns

In this chapter, sample usage patterns are listed with codes. All output is based on the sample data file provided with the setup and variable list is shown below.



Name	Size	Date/Time	Notes	Labels
g1.raw.Block0	55 x 48	8/3/2009 6:15:02 PM	Saved from raw data	RAW
g1.raw.Block1	139 x 48	8/3/2009 6:15:02 PM	Saved from raw data	RAW
g1.raw.Time0	55 x 1	8/3/2009 6:15:02 PM	Time for saved raw data	RAW
g1.raw.Time1	139 x 1	8/3/2009 6:15:02 PM	Time for saved raw data	RAW
g1.oxy.Block0	55 x 16	8/3/2009 6:15:39 PM	Saved from fNIR data	OXY
g1.oxy.Block1	139 x 16	8/3/2009 6:15:39 PM	Saved from fNIR data	OXY
g1.oxy.Time0	55 x 1	8/3/2009 6:15:39 PM	Time for saved fNIR data	OXY
g1.oxy.Time1	139 x 1	8/3/2009 6:15:39 PM	Time for saved fNIR data	OXY
g1.hb.Block0	55 x 16	8/3/2009 6:15:39 PM	Saved from fNIR data	OXY
g1.hb.Block1	139 x 16	8/3/2009 6:15:39 PM	Saved from fNIR data	OXY
g1.hb.Time0	55 x 1	8/3/2009 6:15:39 PM	Time for saved fNIR data	OXY
g1.hb.Time1	139 x 1	8/3/2009 6:15:39 PM	Time for saved fNIR data	OXY
g1.hbo2.Block0	55 x 16	8/3/2009 6:15:39 PM	Saved from fNIR data	OXY
g1.hbo2.Block1	139 x 16	8/3/2009 6:15:39 PM	Saved from fNIR data	OXY
g1.hbo2.Time0	55 x 1	8/3/2009 6:15:39 PM	Time for saved fNIR data	OXY
g1.hbo2.Time1	139 x 1	8/3/2009 6:15:39 PM	Time for saved fNIR data	OXY

Using find command

A sub-list of variables can be easily selected by using 'find' command. Find command can use name and/or label information to select variables. Names/Labels are selected by regular expressions (RegEx). An optional parameter 'verbose' can be added to print out a summary at command output pane.

```
find( "g1.hbo2" verbose);
```

```
>> find( "g1.hbo2" verbose);
   g1.hbo2.Block0      OXY
   g1.hbo2.Block1      OXY
   g1.hbo2.Time0       OXY
   g1.hbo2.Time1       OXY
   find: Found 4 matching variables
```

```
find( "g1.hbo2.B" verbose);
```

```
>> find( "g1.hbo2.B" verbose);
   g1.hbo2.Block0      OXY
   g1.hbo2.Block1      OXY
   find: Found 2 matching variables
```

```
find( "Block0" verbose);
```

```
>> find( "Block0" verbose);
   g1.raw.Block0       RAW
   g1.oxy.Block0       OXY
   g1.hb.Block0        OXY
   g1.hbo2.Block0      OXY
   find: Found 4 matching variables
```

More information about regular expression syntax can be found elsewhere[1].

Similar to using names, labels can be used for variable selection.

```
find( "" "RAW" verbose);
```

```
>> find( "" "RAW" verbose);
   g1.raw.Block0        RAW
   g1.raw.Block1        RAW
   g1.raw.Time0         RAW
   g1.raw.Time1         RAW
   find: Found 4 matching variables
```

Name and label RegEx templates can be used together to further fine-tune the list

```
find( "B" "RAW" verbose);
```

```
>> find( "B" "RAW" verbose);
   g1.raw.Block0        RAW
   g1.raw.Block1        RAW
   find: Found 2 matching variables
```

TIP

Use '**view**' command along with find command to display all variables together.

Executing view find("oxy.Block") will display the following window:

Calculating Averages

A common operation is to calculate mean values for the blocks. There are multiple ways to perform the same operation and their advantages vary depending on the condition. In this section, let's calculate the mean value across time and voxels, for g1.oxy.Block0 and g1.oxy.Block1

[1] http://en.wikipedia.org/wiki/Regular_expression

The following two methods do the same thing in two different ways. The first methods simply uses find function to get all variables and perform the calculation in one line. The second method uses iterative approach and

Method 1

```
result1 = spatialmeanacross( temporalmeanacross( find( "oxy.B" verbose) ) )
```

```
Executing file C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\ave1.fss
    g1.oxy.Block0        OXY
    g1.oxy.Block1        OXY
    find: Found 2 matching variables
    result1 =
    0.262793966249993
```

Method 2

```
delete allData

loop $i from 0 to 1
{
allData = append( @allData temporalmeanwithin( g1.oxy.Block$i));
}

result2 = spatialmeanwithin( temporalmeanwithin( allData) )
```

```
Executing file C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\ave2.fss
    allData is deleted.
    result2 =
    0.262793966249993
```

Note that, in the loop, we have used '@' prefix in-front of allData, which relieves us from initializing the variable before iteration. It merely, disregards the variable if it is the first iteration. Otherwise, we would need to create vector of the same size (columns) and append all other variables and trim the first row (initial all zero). The '@' prefix saved us from all these additional work.

TIP

All commands that end with "**within**" means the operation is performed for each input variable. So it creates the same number of output variables as the number of input variables.

All commands that end with "**across**" means the operation is performed using all input variables at once and creates only one output variables with any number of input variables (with a few exceptions).

Load/Save Variables

fS saves all variables to a fsd data file. These files can be loaded but using “load” command with followed by filename (relative or absolute). All the following commands load the same file.

```
load data0.fsd
```

```
load "data0.fsd"
```

```
load "C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\data0.fsd"
```

```
>> load "data0.fsd"
   Loaded 16 variables from C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\data0.fsd
```

A brief report (file name and number of variables loaded) appear on the command output pane.

Similarly, all current variables can be saved to a file by using save command. The first parameter is the new file name.

```
save data1.fsd
```

```
save "data1.fsd"
```

In addition, you can only save a subset of variables instead of variables all current variables. To do that, add the names of the variables to save after the file name.

```
save "data1.fsd" g1.oxy.Block0 g1.oxy.Block1
```

```
save "data1.fsd" find("oxy.B")
```

After executing the “save” command, a brief report (file name and number of variables saved) will appear on the command output pane.

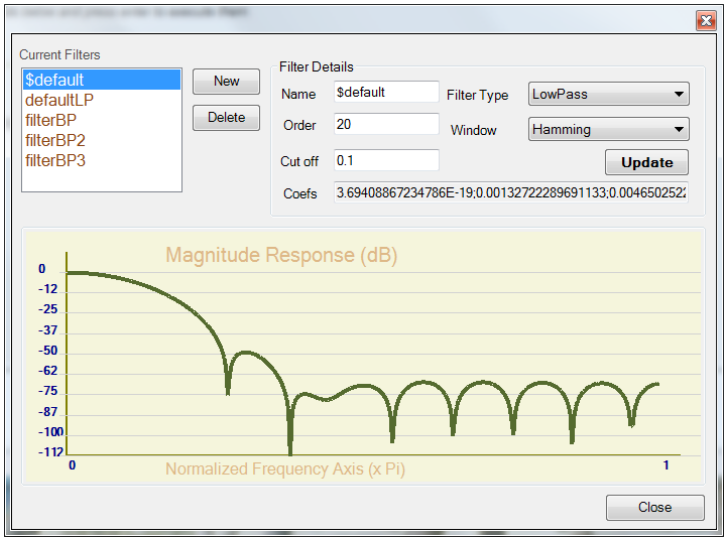
```
>> save "data1.fsd" find("oxy.B")
   Saved 2 variables to C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\data1.fsd
```

Applying FIR Filter

Use filter command to apply FIR filter to select variables. The syntax is as follows

filter <filter_name> <variable name(s)>

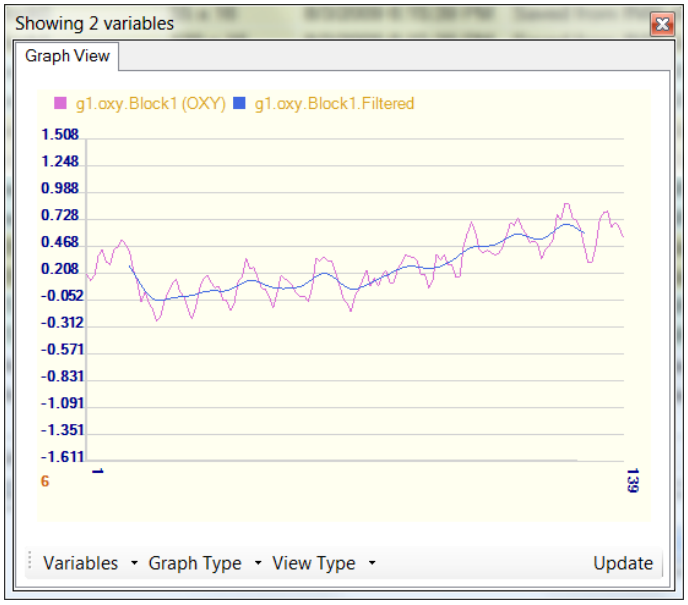
To see the available filter names or design new filters type "**filters**", a new window will appear as shown below.



Here's the code for applying the $default (which is a low-pass FIR filter) to the 2 variables

```
g1.oxy.Block#.Filtered = filter( $default find("oxy.B" verbose) )
```

Two new variables are created "g1.oxy.Block0.Filtered" and "g1.oxy.Block1.Filtered". Below is graphing just voxel4 for g1.oxy.Block1 and its filtered version "g1.oxy.Block1.Filtered"



TIP

Use '**#**' operator to define indexing within names. If it is not used, by default indexing numbers are added such as <name>_0, <name>_1. However, you can define where the numbers should be added by placing '#' within the name

Calculating Oxygenation

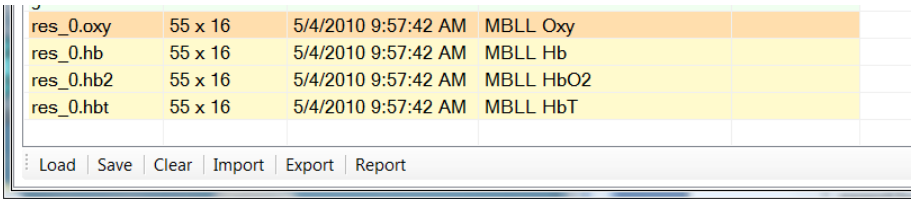
Use mbll command to apply modified beer lambert law (MBLL) to input raw variables (48 column each). Usage is as follows:

mbll(var1 [var2] "baseline" var1 [var2])

At least one input variable is necessary but there's no limit for input variables. Baseline variables can be specified, by using "baseline" keyword before baseline variables. If no baseline variables are used, first 20 samples (with valid data) is used within each input variable.

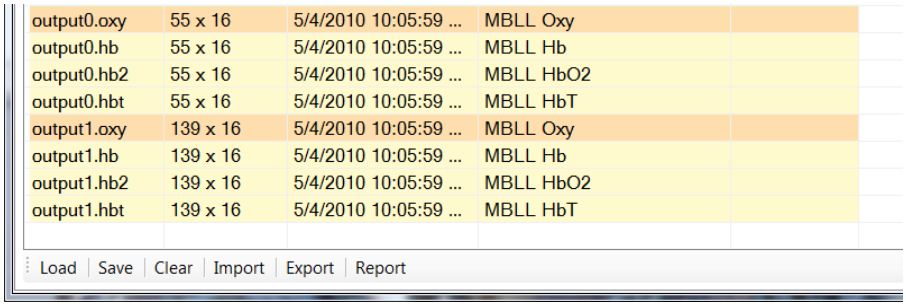
The following line uses the variable g1.raw.Block0 and applies MBLL to create 4 variables; oxygenation (oxy), oxy-Hb (hb2), deoxy-Hb (hb) and total-Hb (hbt) as follows.

```
res = mbll( g1.raw.Block0);
```


res_0.oxy 55 x 16 5/4/2010 9:57:42 AM MBLL Oxy
res_0.hb 55 x 16 5/4/2010 9:57:42 AM MBLL Hb
res_0.hb2 55 x 16 5/4/2010 9:57:42 AM MBLL HbO2
res_0.hbt 55 x 16 5/4/2010 9:57:42 AM MBLL HbT
Load Save Clear Import Export Report

The following command applies MBLL to two raw variables (g1.raw.Block0 and g1.raw.Block1) by using beginning 20 samples of each variable as baseline. A total of 8 variables are created, 4 for each input.

```
output# = mbll( find( "raw.B" verbose ));
```


output0.oxy 55 x 16 5/4/2010 10:05:59 ... MBLL Oxy
output0.hb 55 x 16 5/4/2010 10:05:59 ... MBLL Hb
output0.hb2 55 x 16 5/4/2010 10:05:59 ... MBLL HbO2
output0.hbt 55 x 16 5/4/2010 10:05:59 ... MBLL HbT
output1.oxy 139 x 16 5/4/2010 10:05:59 ... MBLL Oxy
output1.hb 139 x 16 5/4/2010 10:05:59 ... MBLL Hb
output1.hb2 139 x 16 5/4/2010 10:05:59 ... MBLL HbO2
output1.hbt 139 x 16 5/4/2010 10:05:59 ... MBLL HbT
Load Save Clear Import Export Report

Standardizing

Z-scores are one method of normalization.

The standard z-score is defined as follows:

$$z = \frac{x-\mu}{\sigma}$$

where x is a raw value to be standardized, μ is the mean of the dataset where x came from, and σ is the standard deviation of the same dataset.

Use TemporalZScoresAcross command to calculate z-scores across input variables. In the following sample, two variables (g1.oxy.Block0 and g1.oxy.Block1) are used as input

```
res = temporalzscoresacross( find("oxy.B" verbose));
```

```
>> res = temporalzscoresacross( find("oxy.B" verbose));
   g1.oxy.Block0        OXY
   g1.oxy.Block1        OXY
   find: Found 2 matching variables
```

res_0	55 x 16	5/4/2010 10:17:53 ...	Temporal Z-Scores Across
res_1	139 x 16	5/4/2010 10:17:53 ...	Temporal Z-Scores Across

Load | Save | Clear | Import | Export | Report

Splitting Variables

Use split command to create smaller output variables from input variables. Splitting can be done either by specifying the number of output variables (each with equal size that depends on the size of the input variables) or by specifying the size of the output variables and the number of outputs depend on the size of the input variables.

Usage is as follows:

outputs = split(splitDefinition inputVar1 [inputVar2] {inputVar3] …);

splitdefinition is a vector:

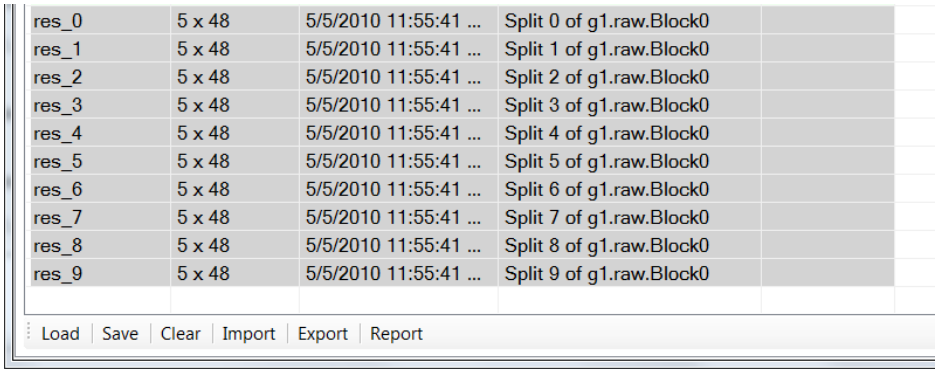
For Fixed Number of Output Blocks: [(Using-Row-Number) (Using-Col-Number)]

For Fixed Length of Output Blocks: [0 0 (Using-Row-Number) (Using-Col-Number)]

The following example creates 10 blocks out of g1.raw.Block0. Input block has 55 rows so each output block has the rounded factor that is 5 rows. Hence, the last rows has been eliminated for the split. To

change that, you can use temporaltrimtop or temporaltrimlast commands to eliminate as much rows as required.

```
res = split( [10 ] g1.raw.Block0 );
```



res_0	5 x 48	5/5/2010 11:55:41 ...	Split 0 of g1.raw.Block0
res_1	5 x 48	5/5/2010 11:55:41 ...	Split 1 of g1.raw.Block0
res_2	5 x 48	5/5/2010 11:55:41 ...	Split 2 of g1.raw.Block0
res_3	5 x 48	5/5/2010 11:55:41 ...	Split 3 of g1.raw.Block0
res_4	5 x 48	5/5/2010 11:55:41 ...	Split 4 of g1.raw.Block0
res_5	5 x 48	5/5/2010 11:55:41 ...	Split 5 of g1.raw.Block0
res_6	5 x 48	5/5/2010 11:55:41 ...	Split 6 of g1.raw.Block0
res_7	5 x 48	5/5/2010 11:55:41 ...	Split 7 of g1.raw.Block0
res_8	5 x 48	5/5/2010 11:55:41 ...	Split 8 of g1.raw.Block0
res_9	5 x 48	5/5/2010 11:55:41 ...	Split 9 of g1.raw.Block0

Load | Save | Clear | Import | Export | Report

Exporting variables to Matlab

Use export2m command to save select variables to a single matlab file. The filename should have an extentions ".m" followed by a list of variable names.

export2m <filename>.m <variable1> <variable2> …

```
export2m data0.m g1.oxy.Block0 g1.oxy.Block1
```

```
>> export2m data0.m g1.oxy.Block0 g1.oxy.Block1
   Exported 2 variables to C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\data0.m
```

Execution of the above command created a new file "data0.m" that when executed by Matlab, loads the two selected variable.

Exporting variables to Text Files

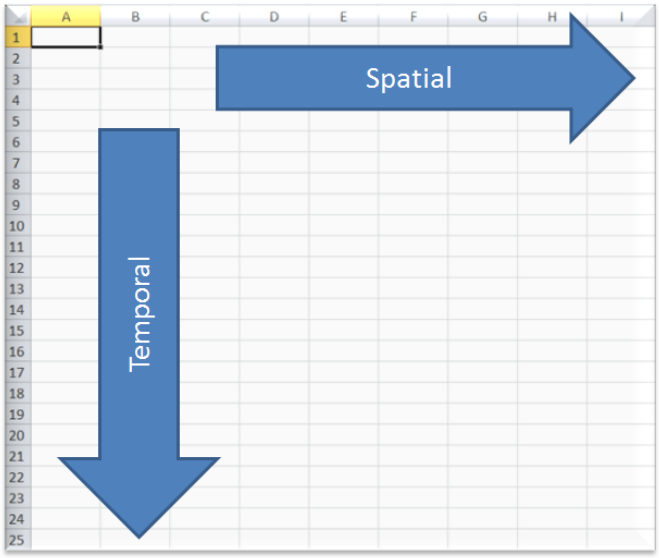
Use export2txt command to save select variables to a single tab separated file that can be opened by many data processing applications. The selected filename should have an extension ".txt".

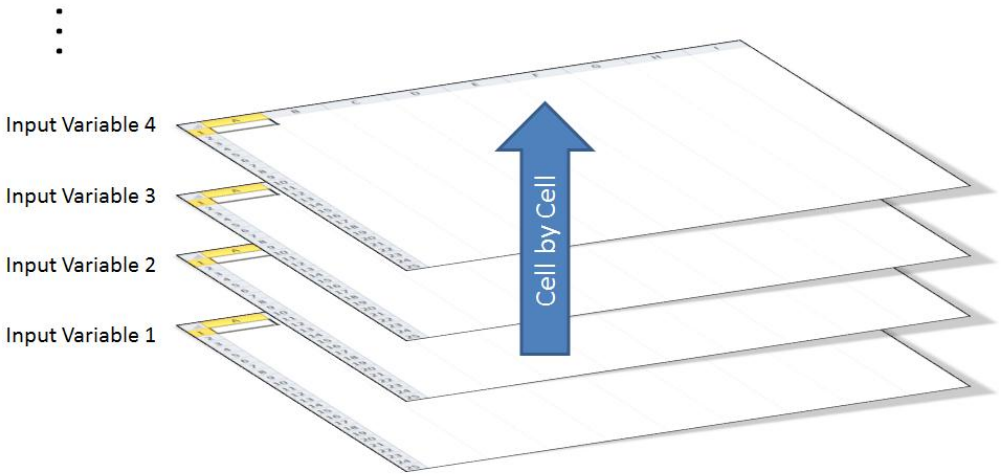
```
export2txt data0.txt g1.oxy.Block0 g1.oxy.Block1
```

```
>> export2txt data0.txt g1.oxy.Block0 g1.oxy.Block1
   Exported 2 variables to C:\Users\Hasan7\Documents\ACA\Data\TEST_DATA\data0.txt
```

Naming Convention of Commands

Command names indicate certain features of the commands. If the command name starts with either 'Temporal', 'Spatial' or 'Cell by Cell', this specify the direction of the operation. For example, TemporalMeanWithin command, calculates means of all columns (of input variable) and the output variable has the same number of columns with the input variable. Similarly, SpatialMeanWithin variable calculates means of all rows (of input variable) and the output variable has the same number of rows with the input variable. 'Cell by Cell' specifies use of each cell separately across multiple input variables. Below is an illustration of the direction of commands.





If the command name ends with either 'within', this indicates operation is performed on each input variable separately and if command name ends with 'across', this indicates operation is performed on all input variables together. For example, TemporalMeanWithin calculates means on each input variable separately, and TemporalMeanAcross calculates one global mean from all input variables.

List of Commands

Here's list of current variables classified by the type. To get more information about these command, type their name in the command prompt and hit enter. Description and usage will appear in the command output pane.

Math/Arithmetic

abs
acos
add
asin
atan
base
ceiling
cos
cosh
divide
exp
floor
log
ln
log10
multiply
random
randominteger
round
sin
sinh
sqrt
subtract
tan
truncate

Utilities/Tools

about
admin
cd
chan2vox
clear
close
delete
dir
display
echo

edit
editor
exit
filters
export2img
export2m
export2txt
help
hide
history
import
importdata
mbll
list
load
new
open
pwd
rename
renamelabel
report
reset
save
variables
version
view
vox2chan
vox2chans

Statistical Comparison

ttest
ttestpaired

Signal Processing - Temporal

append
appendiflabelsmatch
appendtemporalmeans
appendtemporalmeansiflabelsmatch
appendcolumns
getcolumns
getrows
rejectcolumns
sortwithin
temporalconfidenceintervalwithin
temporalcorrwithin
temporalcountwithin

temporaldecomposewithin
temporaldllacross
temporaldulacross
temporaleffectsizedacross
temporaleffectsizeracross
temporalfixmissingacross
temporalfixmissingwithin
temporalmeanacross
temporalmeanwithin
temporalmaxacross
temporalmaxwithin
temporalmedianwithin
temporalminacross
temporalminwithin
temporalpeakindexwithin
temporalrllacross
temporalrulacross
temporalsamplestdacross
temporalsamplestdwithin
temporalstdacross
temporalstdwithin
temporalstderrwithin
temporalslopewithin
temporalsizeacross
temporalsizewithin
temporalskewnesswithin
temporaltrimfirst
temporaltrimlast
temporaltrimnanwithin
temporaltrimnanpivotwithin
temporalzscoreswithin
temporalzscoresacross

Signal Processing - Spatial

spatialfixmissingwithin
spatialmaxacross
spatialmaxwithin
spatialmeanacross
spatialmeanwithin
spatialminacross
spatialminwithin
spatialsamplestdwithin
spatialstdacross
spatialstdwithin
spatialtrimfirst
spatialtrimlast

spatialslopewithin
spatialsizewithin

Signal Processing - Cell by Cell

cellbycellmeanacross
cellbycellstdacross
cellbycellstderracross

Signal Processing - Misc/Tools

addlabel
cbsi
defineblocktimes
extractwithtime
extractwithindex
getblocktimes
getfilenames
filter
find
findcount
gamma
getsamesized
loggamma
match
medianfilter
removealllabels
smar
split
tostring
tovar